# match and patterns

Rust basics · match and patterns

## In PHP

`match` is an expression and compares with `===`. A missing arm throws `UnhandledMatchError` when that line runs. `??` is the stand-in for a missing value.

A function that takes `?int $age` handles both cases, the same way the Rust function below handles `Some` and `None`.

*age.php*

```
<?php

function show_age(string $name, ?int $age): void
{
    if ($age === null) {
        echo "$name has no age set\n";
        return;
    }

    echo "$name is $age\n";
}

show_age("Ada", 36);
show_age("Bea", null);

$years = null;
echo $years ?? 0, "\n";
```

This `match` has no arm for `3`:

*label.php*

```
<?php

function label(int $n): string
{
    return match ($n) {
        1 => "one",
        2 => "two",
    };
}
```

```
echo label(3), "\n";
```

```
 Fatal error: Uncaught UnhandledMatchError: Unhandled match case 3
```

## In Rust

`match` is an expression. Every value of the type needs an arm, or a `_` arm. Literals, `|`, and ranges are patterns. `3..=5` includes both ends. `6..10` excludes `10`. Exclusive ranges in a `match` have been stable since 1.80. `n @ 10..=20` binds the value that matched. `(x, y)` takes a tuple apart.

`&'static str` means the text is baked into the binary for the whole program. Lifetimes, including why `'static` is special, are chapter 3. For now, treat it as "a string slice that lives forever".

*src/main.rs*

```
 fn label(n: i32) -> &'static str {
    match n {
        0 => "zero",
        1 | 2 => "small",
        3..=5 => "mid",
        6..10 => "high",
        _ => "other",
    }
}

fn describe(point: (i32, i32)) -> String {
    match point {
        (0, 0) => "origin".to_string(),
        (x, y) => format!("{x},{y}"),
    }
}

fn main() {
    let kind = label(4);
    let high = label(7);
    let other = label(12);
    println!("{kind} {high} {other}");

    let origin = describe((0, 0));
    let moved = describe((2, 3));
    println!("{origin} {moved}");

    let band = match 12 {
        n @ 10..=20 => format!("in band {n}"),
        other => format!("outside {other}"),
```

```
    };
    println!("{band}");
}
```

`label` prints `mid high other`. `6..10` is `6`, `7`, `8`, and `9`, so `10` would be `other`. The tuple prints `origin` and `2,3`. The binding prints `in band 12`.

Drop the `_` arm and the program does not compile. This is `E0004` for the arms above:

```
 error[E0004]: non-exhaustive patterns: `i32::MIN..=-1_i32` and
`10_i32..=i32::MAX` not covered
 --> src/main.rs:2:11
  |
2 |     match n {
  |           ^ patterns `i32::MIN..=-1_i32` and `10_i32..=i32::MAX` not
covered
  |
  = note: the matched value is of type `i32`
help: ensure that all possible cases are being handled by adding a match arm
with a wildcard pattern, a match arm with multiple or-patterns as shown, or
multiple match arms
```

A guard is `if` after the pattern. The arm runs only when the pattern matches and the condition is true.

`Option` is `Some(value)` or `None`. `match` handles both. `if let Some(years) = maybe` handles the `Some` arm and skips the rest. `let Some(years) = age else { return; }` is the shape the chapter 1 calculator uses: leave the function when the pattern does not match. `unwrap_or(0)` is PHP's `??`.

*src/main.rs*

```
 fn age_or_zero(age: Option<i32>) -> i32 {
    age.unwrap_or(0)
}

fn show_age(name: &str, age: Option<i32>) -> String {
    match age {
        Some(years) => format!("{name} is {years}"),
        None => format!("{name} has no age set"),
    }
}

fn main() {
    let named = show_age("Ada", Some(36));
    let missing = show_age("Bea", None);
    println!("{named}");
    println!("{missing}");
```

```
    let maybe = Some(36);
    if let Some(years) = maybe {
        println!("if let {years}");
    }

    let age = Some(36);
    let Some(years) = age else {
        return;
    };
    println!("else {years}");

    let fallback = age_or_zero(None);
    println!("{fallback}");

    let size = match 150 {
        n if n >= 100 => "large",
        _ => "small",
    };
    println!("{size}");
}
```

`show_age` prints `Ada is 36` and `Bea has no age set`. `if let` prints `if let 36`. `let ... else` prints `else 36`. `unwrap_or` prints `0`. The guard prints `large`.

> **Watch for:** A missing arm is a compile error in Rust. PHP throws `UnhandledMatchError` when that line runs. A guard comes after the pattern, and only then.

> **Idiomatic Rust:** `_` when the remaining values share one result. `match` on `Option` when both arms matter. `let ... else` when `None` should leave the function. `unwrap_or` for a default.

## Side by side

| | PHP | Rust |
|---|---|---|
| Expression | `match` | `match` |
| Missing arm | `UnhandledMatchError` when the line runs | `E0004` at compile time |

| | PHP | Rust |
|---|---|---|
| Several values | `1, 2` | `1 \| 2` |
| Range | not a pattern | `3..=5` and `6..10` |
| Optional age | `?int` and `=== null` | `Option`, `Some` / `None` |
| Default | `??` | `unwrap_or` |

## Try it

Open the second `src/main.rs` (the Option program) in the Playground. It prints `large`. Change `150` to `40` and run it again. The guard does not match, so the line is `small`.

## Key takeaways

- `match` is an expression. Arms can be literals, `|`, `..=`, `..` (stable since 1.80), a binding, or a tuple.
- A guard is `if` after the pattern. The arm runs only when the pattern and the condition both hold.
- A missing arm is `E0004` and the program does not compile. PHP throws `UnhandledMatchError` when the line runs.
- `match` on `Some` and `None`, then `if let`, then `let ... else`. `unwrap_or` is `??`.

What's next: Input classifier